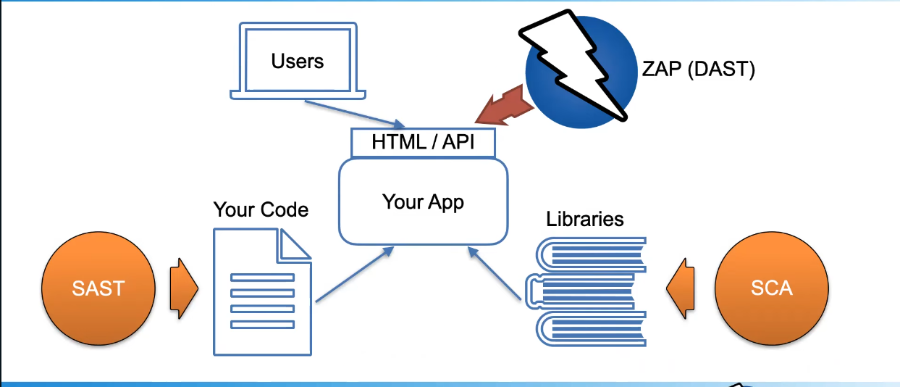
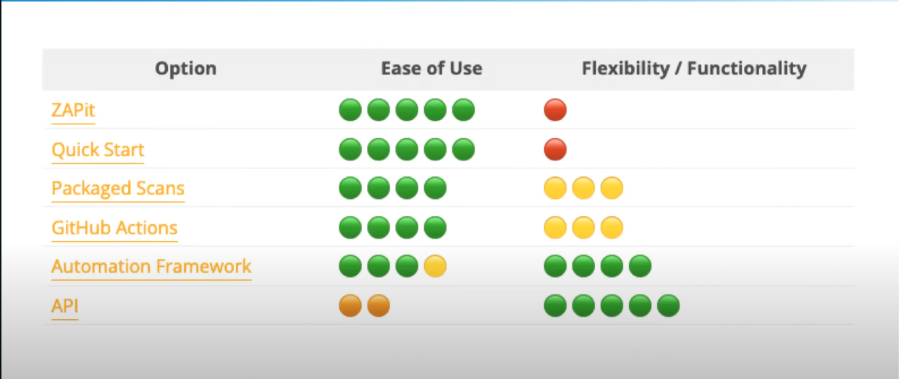
API Security R&D

* Zap is a DAST tool used for finding vulnerabilties in a running application.
* It works more like a penteration testing.
* It is a cross platform.
* It can run as desktop, cmdline, daemon, docker, Github Actions.
* SAST:-
  + Actually sast tools like bearer looks into the code and analyze the code to find potential vulnerabilities.
* SCA(Software compostion analysis):
  + This analysis libraries we used in the code, and looking for known libraries used
* Actually sast and sca analyze the components that make up our application, but they are not actually attacking them, they are just looking at the contents of how the application is built.
* Whereas ZAP works differently, it actively attacks the application in the same way malicious attacker might
* 
* Use case of ZAP
* First we need to authenticate it to the target application. We can do this manually by logging in via browser proxy through zap or we can configure zap to handle the authentication.
* Then zap will explore the application
* Zap has three spiders all of which crawl the application in different ways and have different strengths and weaknesses.
* We can also import wide range if api definitions so zap can really understand exactly what APIs your application provides.
* As we are exporing the application ZAP automatically does passive scannig. So it looks for requests and responses. And can actually identifies a set of vulnerabilties just on that basis.(passive scanning)
* But those aren’t the serious vulnerabities it can find. The serious ones are find via attacking. So this is where ZAP acually attacks the application, sends potentially malicious payloads.
* At last we can generate a report like html.
* **Automate options**
* 
* ZAP can run in **headless mode via Docker** for CI/CD pipelines.
* Supports **ZAP CLI** and **REST API** for scripting.
* Can be integrated into SaaS security platforms or AWS marketplace solutions.
* **Features list of zaproxy**

Core proxy & crawling

* **Intercepting proxy** (HTTP/HTTPS) — inspect/modify traffic in-flight.
* **HTTPS/TLS support** — dynamic ZAP root CA, client cert (mTLS) support.
* **Traditional Spider** — link crawling from HTML.
* **AJAX Spider** — headless browser-based crawling for JS SPAs.
* **Forced Browse** (directory brute-force) — discover hidden endpoints/files.
* **Manual Explore / Requester** — craft requests, resend, and replay.

# **Scanning & testing**

* **Passive scanner** — safe checks on observed traffic (no active attacks).
* **Active scanner** — intrusive tests (SQLi, XSS, File inclusion, etc.) with configurable rules and scan policies.
* **Fuzzer** — targeted fuzzing of parameters/headers.
* **Forced browsing + fuzzing combos** — discover and test guessed endpoints.
* **WebSocket scanning** — intercept and fuzz WebSocket messages.

# **Authentication & session handling**

* **Contexts** — group of URLs with shared settings (scope, auth, session mgmt).
* **Auth types:** form-based, HTTP Basic/Digest/NTLM, script-based authentication, OAuth/OIDC flows (via scripts/helper flows).
* **Users** — create multiple users per context (credentials) and scan as a user.
* **Forced User Mode** — force requests to use a specific user session.
* **Session Management** — cookie / token handling, session checks, detect logged-in/out state.

# **Automation & integration**

* **REST API (fully scriptable)** — control everything (contexts, users, spider, scans, reports).
* **CLI & headless runs** — run ZAP without UI (good for CI).
* **Docker images** — official images for CI/CD pipelines.
* **Automation Framework (YAML)** — predefine plans (auth, spider, scan, report) and run them headless.
* **OpenAPI / Swagger import** — scan API definitions.
* **IDE / CI-friendly outputs** — JSON, XML, HTML reports; can be parsed by pipelines.

# **Extensibility & scripting**

* **ZAP Marketplace / Add-ons** — install additional scanners, tools, integrations.
* **Script support** — JavaScript, Jython (Python), Zest, Ruby, etc., for custom scanners, auth, or workflow.
* **Zest scripting** — recordable security test scripts.
* **Extendable active/passive rules** — add or tune rules and thresholds.

# **Usability & developer helpers**

* **HUD (Heads Up Display)** — in-browser UI overlay to work with ZAP while browsing the app (ideal for manual testing).
* **Launch Browser** — built-in browser already proxied & trusting ZAP cert.
* **History & Request/Response Inspector** — search, compare, and export traffic.
* **Alert management** — triage, risk levels, evidence, CWE/CVSS IDs.
* **Report generation** — custom HTML, XML, JSON reports and templates.

# **Advanced & enterprise-friendly features**

* **Scan policies & attack strength tuning** — customize which rules run and their aggressiveness.
* **Rate limiting & politeness** — schedule or throttle scans to avoid DoS.
* **Context export/import** — re-use configurations across environments or share with customers.
* **Scheduling & headless automation** — integrate into platform pipelines and multi-tenant flows.
* **Scripting for SSO and complex auth** — support captive SSO, multi-step logins, CSRF handling.

# **Security & safety controls**

* **Allow/deny lists (scope)** — limit what gets scanned / crawled.
* **Active scan safeguards** — disable specific invasive rules or tune to reduce impact.
* **User permission guidance** — run scans using least-privilege test accounts; session-only scans.

**My progress on this task**

* At first i understand what is the zaproxy tool and why it is used and how it differ from sast tools like bearer.
* After that i cloned the zaproxy repo started building it, but the ui did’nt open because intially it is asking username and password.
* On doing some reasearch i got to know how to open the ui
* Later i created a repo in my organization with all zaproxy files added it. So after that on doing research i got to know in ui we have, different scans
  + Automated scan
  + Manual explorer
* The main difference between these both are for automated scan intially we can our testing and staging env url with http access. In below we can see the traditional spider and ajax spider.
* Main difference between both of them are
  + The traditional spider is a crawler that parses the **HTML of web pages** (the DOM at load time, links in <a href=...>, forms, etc.) to discover new URLs.
  + The AJAX spider uses a **browser engine (through Selenium + Crawljax)** to render pages **like a real user**. It executes JavaScript, clicks links, interacts with the DOM, and captures dynamically loaded URLs.
* So later on running a scan depending on the spider the scan results will be excuted.
* The results will be divided based on risk level and alerts.
* So later we dicided to implement the automated scan feature into secops. So first i builded a docker image from the repo with flags included to the automated scan.
* Later i tested the docker image with a sample https urls too and http, the scanner performed well for https urls which doesnt have any authentication, but with the urls that have authentication the docker image only scanned the login page when the authentication provided in the docker flags also.
* So then we started doing research more how can we achieve, if we compare the desktop appliation of zaproxy , for doing this https with form-based authentication there are several things to do.
* Now for version 1 we are thinking to give no authentication and http scanning so later we can go for basic auth and oauth.